|  |  |
| --- | --- |
| ACTIVIDAD 3  Programación orientada a objetos | uNIVERSIDAD NACIONAL DE COLOMBIA  Profesor: Walter Arboleda  Daniel Felipe Zapata Patrón  Cc: 1000406800 |

PARTE 1

Código fuente: https://github.com/danielzp2000/actividad3PooUnalMed/tree/660e38e25dcc489a73b455f4fb8b64a8c92cb91f/Parte%201

Capítlo 3: Estructura secuencial

Ejercicios propuestos

* 18. Se tiene la siguiente información de un empleado:
* Código del empleado
* Nombres
* Número de horas trabajadas al mes
* Valor hora trabajada
* Porcentaje de retención en la fuente.

Haga un algoritmo que muestre: código, nombres, salario bruto y salario neto.

Código:

class SalarioEmpleado:

def \_\_init\_\_(self, horas, tarifa, retencion):

self.horas\_trabajadas = horas

self.tarifa\_hora = tarifa

self.porcentaje\_retencion = retencion

def calcular\_salario\_bruto(self):

bruto = self.horas\_trabajadas \* self.tarifa\_hora

return bruto

def calcular\_salario\_neto(self):

bruto = self.calcular\_salario\_bruto()

neto = bruto - (self.porcentaje\_retencion \* bruto)

return neto

Interfaz gráfica:

import tkinter as tk

from tkinter import messagebox

class VentanaPrincipal(tk.Tk):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.title("Cálculo de Salarios")

# Campos de entrada

tk.Label(self, text="Código Empleado:").grid(row=0, column=0, padx=10, pady=5)

self.txt\_codigo\_empleado = tk.Entry(self)

self.txt\_codigo\_empleado.grid(row=0, column=1, padx=10, pady=5)

tk.Label(self, text="Nombre:").grid(row=1, column=0, padx=10, pady=5)

self.txt\_nombre = tk.Entry(self)

self.txt\_nombre.grid(row=1, column=1, padx=10, pady=5)

tk.Label(self, text="Horas Trabajadas:").grid(row=2, column=0, padx=10, pady=5)

self.txt\_horas\_trabajadas = tk.Entry(self)

self.txt\_horas\_trabajadas.grid(row=2, column=1, padx=10, pady=5)

tk.Label(self, text="Tarifa por Hora:").grid(row=3, column=0, padx=10, pady=5)

self.txt\_tarifa\_hora = tk.Entry(self)

self.txt\_tarifa\_hora.grid(row=3, column=1, padx=10, pady=5)

tk.Label(self, text="Retención Fuente (%):").grid(row=4, column=0, padx=10, pady=5)

self.txt\_retencion\_fuente = tk.Entry(self)

self.txt\_retencion\_fuente.grid(row=4, column=1, padx=10, pady=5)

# Botones

tk.Button(self, text="Calcular", command=self.calcular).grid(row=5, column=0, padx=10, pady=10)

tk.Button(self, text="Borrar", command=self.borrar).grid(row=5, column=1, padx=10, pady=10)

tk.Button(self, text="Salir", command=self.salir).grid(row=6, column=0, columnspan=2, padx=10, pady=10)

# Campos de salida

tk.Label(self, text="Código:").grid(row=7, column=0, padx=10, pady=5)

self.txt\_codigo = tk.Entry(self, state="readonly")

self.txt\_codigo.grid(row=7, column=1, padx=10, pady=5)

tk.Label(self, text="Nombre:").grid(row=8, column=0, padx=10, pady=5)

self.txt\_nombre\_dos = tk.Entry(self, state="readonly")

self.txt\_nombre\_dos.grid(row=8, column=1, padx=10, pady=5)

tk.Label(self, text="Salario Bruto:").grid(row=9, column=0, padx=10, pady=5)

self.txt\_salario\_bruto = tk.Entry(self, state="readonly")

self.txt\_salario\_bruto.grid(row=9, column=1, padx=10, pady=5)

tk.Label(self, text="Salario Neto:").grid(row=10, column=0, padx=10, pady=5)

self.txt\_salario\_neto = tk.Entry(self, state="readonly")

self.txt\_salario\_neto.grid(row=10, column=1, padx=10, pady=5)

# Método para calcular los salarios

def calcular(self):

try:

codigo = self.txt\_codigo\_empleado.get()

nombre = self.txt\_nombre.get()

horas = float(self.txt\_horas\_trabajadas.get())

tarifa = float(self.txt\_tarifa\_hora.get())

retencion = float(self.txt\_retencion\_fuente.get()) / 100

empleado = SalarioEmpleado(horas, tarifa, retencion)

salario\_bruto = empleado.calcular\_salario\_bruto()

salario\_neto = empleado.calcular\_salario\_neto()

# Mostrar resultados en los campos de salida

self.txt\_codigo.config(state="normal")

self.txt\_codigo.delete(0, tk.END)

self.txt\_codigo.insert(0, codigo)

self.txt\_codigo.config(state="readonly")

self.txt\_nombre\_dos.config(state="normal")

self.txt\_nombre\_dos.delete(0, tk.END)

self.txt\_nombre\_dos.insert(0, nombre)

self.txt\_nombre\_dos.config(state="readonly")

self.txt\_salario\_bruto.config(state="normal")

self.txt\_salario\_bruto.delete(0, tk.END)

self.txt\_salario\_bruto.insert(0, f"{salario\_bruto:.2f}")

self.txt\_salario\_bruto.config(state="readonly")

self.txt\_salario\_neto.config(state="normal")

self.txt\_salario\_neto.delete(0, tk.END)

self.txt\_salario\_neto.insert(0, f"{salario\_neto:.2f}")

self.txt\_salario\_neto.config(state="readonly")

except ValueError:

messagebox.showerror("Error", "Por favor ingresa valores válidos.")

# Método para borrar los campos de entrada y salida

def borrar(self):

self.txt\_codigo\_empleado.delete(0, tk.END)

self.txt\_nombre.delete(0, tk.END)

self.txt\_horas\_trabajadas.delete(0, tk.END)

self.txt\_tarifa\_hora.delete(0, tk.END)

self.txt\_retencion\_fuente.delete(0, tk.END)

self.txt\_codigo.config(state="normal")

self.txt\_codigo.delete(0, tk.END)

self.txt\_codigo.config(state="readonly")

self.txt\_nombre\_dos.config(state="normal")

self.txt\_nombre\_dos.delete(0, tk.END)

self.txt\_nombre\_dos.config(state="readonly")

self.txt\_salario\_bruto.config(state="normal")

self.txt\_salario\_bruto.delete(0, tk.END)

self.txt\_salario\_bruto.config(state="readonly")

self.txt\_salario\_neto.config(state="normal")

self.txt\_salario\_neto.delete(0, tk.END)

self.txt\_salario\_neto.config(state="readonly")

# Método para salir de la aplicación

def salir(self):

self.destroy()

if \_\_name\_\_ == "\_\_main\_\_":

app = VentanaPrincipal()

app.mainloop()

* 19. Dado el valor del lado en un triángulo equilátero, haga un algoritmo que obtenga el perímetro, el valor de la altura y el área del triángulo.

Código:

import math

class Pitagoras:

def \_\_init\_\_(self, longitud):

self.longitud = longitud

def calcular\_ultimo\_lado(self):

medio\_lado = self.longitud / 2

lado = math.sqrt(self.longitud\*\*2 - medio\_lado\*\*2)

return lado

class Operaciones:

def \_\_init\_\_(self, longitud):

self.longitud = longitud

def calcular\_area(self):

pitagoras = Pitagoras(self.longitud)

altura = pitagoras.calcular\_ultimo\_lado()

area = self.longitud \* altura / 2

return area

def calcular\_perimetro(self):

perimetro = self.longitud \* 3

return perimetro

INTERFAZ GRÁFICA:

import tkinter as tk

from tkinter import messagebox

class Ventana(tk.Tk):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.title("Cálculo de Triángulo Equilátero")

# Campo para ingresar la longitud

tk.Label(self, text="Longitud del lado:").grid(row=0, column=0, padx=10, pady=10)

self.txt\_longitud = tk.Entry(self)

self.txt\_longitud.grid(row=0, column=1, padx=10, pady=10)

# Botones

tk.Button(self, text="Calcular", command=self.calcular).grid(row=1, column=0, padx=10, pady=10)

tk.Button(self, text="Borrar", command=self.borrar).grid(row=1, column=1, padx=10, pady=10)

tk.Button(self, text="Salir", command=self.salir).grid(row=2, column=0, columnspan=2, padx=10, pady=10)

# Campos de salida para área y perímetro

tk.Label(self, text="Área:").grid(row=3, column=0, padx=10, pady=10)

self.txt\_area = tk.Entry(self, state="readonly")

self.txt\_area.grid(row=3, column=1, padx=10, pady=10)

tk.Label(self, text="Perímetro:").grid(row=4, column=0, padx=10, pady=10)

self.txt\_perimetro = tk.Entry(self, state="readonly")

self.txt\_perimetro.grid(row=4, column=1, padx=10, pady=10)

def calcular(self):

try:

longitud = float(self.txt\_longitud.get())

operacion = Operaciones(longitud)

area = operacion.calcular\_area()

perimetro = operacion.calcular\_perimetro()

# Mostrar resultados

self.txt\_area.config(state="normal")

self.txt\_area.delete(0, tk.END)

self.txt\_area.insert(0, f"{area:.2f}")

self.txt\_area.config(state="readonly")

self.txt\_perimetro.config(state="normal")

self.txt\_perimetro.delete(0, tk.END)

self.txt\_perimetro.insert(0, f"{perimetro:.2f}")

self.txt\_perimetro.config(state="readonly")

except ValueError:

messagebox.showerror("Error", "Por favor ingresa un valor numérico válido.")

def borrar(self):

self.txt\_longitud.delete(0, tk.END)

self.txt\_area.config(state="normal")

self.txt\_area.delete(0, tk.END)

self.txt\_area.config(state="readonly")

self.txt\_perimetro.config(state="normal")

self.txt\_perimetro.delete(0, tk.END)

self.txt\_perimetro.config(state="readonly")

def salir(self):

self.destroy()

if \_\_name\_\_ == "\_\_main\_\_":

app = Ventana()

app.mainloop()

Capítulo 4: Estructura decisión lógica.

Ejercicios resueltos

* 7. Hacer un algoritmo que, dados dos valores numéricos A y B, escriba un mensaje diciendo si A es mayor, menor o igual a B.

Código:

import tkinter as tk

from tkinter import messagebox

# Clase que realiza la comparación entre dos números

class ComparacionValores:

def \_\_init\_\_(self, numero\_a, numero\_b):

self.numero\_a = numero\_a

self.numero\_b = numero\_b

def determinar\_relacion(self):

if self.numero\_a > self.numero\_b:

return "A es mayor que B"

elif self.numero\_a < self.numero\_b:

return "A es menor que B"

else:

return "A es igual a B"

# Ventana principal con Tkinter

class Ventana(tk.Tk):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.title("Comparación de Valores")

# Etiquetas y campos de entrada

tk.Label(self, text="Número A:").grid(row=0, column=0, padx=10, pady=10)

self.txt\_numero\_a = tk.Entry(self)

self.txt\_numero\_a.grid(row=0, column=1, padx=10, pady=10)

tk.Label(self, text="Número B:").grid(row=1, column=0, padx=10, pady=10)

self.txt\_numero\_b = tk.Entry(self)

self.txt\_numero\_b.grid(row=1, column=1, padx=10, pady=10)

# Botones

tk.Button(self, text="Calcular", command=self.calcular).grid(row=2, column=0, padx=10, pady=10)

tk.Button(self, text="Borrar", command=self.borrar).grid(row=2, column=1, padx=10, pady=10)

tk.Button(self, text="Salir", command=self.salir).grid(row=3, column=0, columnspan=2, padx=10, pady=10)

# Campo para mostrar el resultado

self.txt\_resultado = tk.Entry(self, state="readonly")

self.txt\_resultado.grid(row=4, column=0, columnspan=2, padx=10, pady=10)

# Método para calcular la relación

def calcular(self):

try:

numero\_a = float(self.txt\_numero\_a.get())

numero\_b = float(self.txt\_numero\_b.get())

comparacion = ComparacionValores(numero\_a, numero\_b)

resultado = comparacion.determinar\_relacion()

self.txt\_resultado.config(state="normal")

self.txt\_resultado.delete(0, tk.END)

self.txt\_resultado.insert(0, resultado)

self.txt\_resultado.config(state="readonly")

except ValueError:

messagebox.showerror("Error", "Por favor ingresa números válidos.")

# Método para borrar los campos de texto

def borrar(self):

self.txt\_numero\_a.delete(0, tk.END)

self.txt\_numero\_b.delete(0, tk.END)

self.txt\_resultado.config(state="normal")

self.txt\_resultado.delete(0, tk.END)

self.txt\_resultado.config(state="readonly")

# Método para salir de la aplicación

def salir(self):

self.destroy()

# Punto de entrada principal

if \_\_name\_\_ == "\_\_main\_\_":

app = Ventana()

app.mainloop()

* 10. Cierta universidad para liquidar el pago de matrícula de un estudiante le exige los siguientes datos:
* Número de inscripción
* Nombres
* Patrimonio
* Estrato social

La universidad cobra un valor constante para cada estudiante de $50.000. Si el patrimonio es mayor que $2'000.000 y el estrato superior a 3, se le incrementa un porcentaje del 3% sobre el patrimonio. Hacer un algoritmo que muestre:

* Número de inscripción
* Nombres
* Pago de matrícula

Código:

class CalculoMatricula:

def \_\_init\_\_(self, patrimonio, estrato):

self.patrimonio = patrimonio

self.estrato = estrato

self.costo\_matricula = 50000 # Costo base de la matrícula

def calcular\_pago\_matricula(self):

if self.patrimonio > 2000000 and self.estrato > 3:

self.costo\_matricula += 0.03 \* self.patrimonio

return self.costo\_matricula

import tkinter as tk

from tkinter import messagebox

class Ventana(tk.Tk):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.title("Cálculo de Matrícula")

# Etiquetas y campos de entrada

tk.Label(self, text="Número de Inscripción:").grid(row=0, column=0, padx=10, pady=10)

self.txt\_num\_inscripcion = tk.Entry(self)

self.txt\_num\_inscripcion.grid(row=0, column=1, padx=10, pady=10)

tk.Label(self, text="Nombres:").grid(row=1, column=0, padx=10, pady=10)

self.txt\_nombres = tk.Entry(self)

self.txt\_nombres.grid(row=1, column=1, padx=10, pady=10)

tk.Label(self, text="Patrimonio:").grid(row=2, column=0, padx=10, pady=10)

self.txt\_patrimonio = tk.Entry(self)

self.txt\_patrimonio.grid(row=2, column=1, padx=10, pady=10)

tk.Label(self, text="Estrato:").grid(row=3, column=0, padx=10, pady=10)

self.txt\_estrato = tk.Entry(self)

self.txt\_estrato.grid(row=3, column=1, padx=10, pady=10)

# Botones

tk.Button(self, text="Calcular", command=self.calcular).grid(row=4, column=0, padx=10, pady=10)

tk.Button(self, text="Borrar", command=self.borrar).grid(row=4, column=1, padx=10, pady=10)

tk.Button(self, text="Salir", command=self.salir).grid(row=5, column=0, columnspan=2, padx=10, pady=10)

# Campos de salida

tk.Label(self, text="Resultado:").grid(row=6, column=0, padx=10, pady=10)

self.txt\_resultado = tk.Entry(self, state="readonly")

self.txt\_resultado.grid(row=6, column=1, padx=10, pady=10)

def calcular(self):

try:

num\_inscripcion = self.txt\_num\_inscripcion.get()

nombres = self.txt\_nombres.get()

patrimonio = float(self.txt\_patrimonio.get())

estrato = int(self.txt\_estrato.get())

matricula = CalculoMatricula(patrimonio, estrato)

pago\_matricula = matricula.calcular\_pago\_matricula()

resultado = f"Número: {num\_inscripcion}\nNombre: {nombres}\nPago Matrícula: ${pago\_matricula:.2f}"

self.txt\_resultado.config(state="normal")

self.txt\_resultado.delete(0, tk.END)

self.txt\_resultado.insert(0, f"${pago\_matricula:.2f}")

self.txt\_resultado.config(state="readonly")

except ValueError:

messagebox.showerror("Error", "Por favor ingresa valores válidos para patrimonio y estrato.")

def borrar(self):

self.txt\_num\_inscripcion.delete(0, tk.END)

self.txt\_nombres.delete(0, tk.END)

self.txt\_patrimonio.delete(0, tk.END)

self.txt\_estrato.delete(0, tk.END)

self.txt\_resultado.config(state="normal")

self.txt\_resultado.delete(0, tk.END)

self.txt\_resultado.config(state="readonly")

def salir(self):

self.destroy()

if \_\_name\_\_ == "\_\_main\_\_":

app = Ventana()

app.mainloop()

Ejercicios propuestos

* 22. Elaborar un algoritmo que entre el nombre de un empleado, su salario básico por hora y el número de horas trabajadas en el mes; escriba su nombre y salario mensual si éste es mayor de $450.000, de lo contrario escriba sólo el nombre.

Código:

class CalculoNomina:

def \_\_init\_\_(self, horas\_trabajadas, valor\_hora, nombre):

self.horas\_trabajadas = horas\_trabajadas

self.valor\_hora = valor\_hora

self.nombre = nombre

def calcular\_nomina(self):

salario = self.horas\_trabajadas \* self.valor\_hora

if salario < 450000:

return f"{self.nombre}"

else:

return f"{self.nombre}\n{salario:.2f}"

#INTERFAZ GRÁFICA:

import tkinter as tk

from tkinter import messagebox

class Ventana(tk.Tk):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.title("Cálculo de Nómina")

# Campos de entrada

tk.Label(self, text="Nombre:").grid(row=0, column=0, padx=10, pady=5)

self.txt\_nombre = tk.Entry(self)

self.txt\_nombre.grid(row=0, column=1, padx=10, pady=5)

tk.Label(self, text="Horas Trabajadas:").grid(row=1, column=0, padx=10, pady=5)

self.txt\_horas\_mes = tk.Entry(self)

self.txt\_horas\_mes.grid(row=1, column=1, padx=10, pady=5)

tk.Label(self, text="Valor por Hora:").grid(row=2, column=0, padx=10, pady=5)

self.txt\_valor\_hora = tk.Entry(self)

self.txt\_valor\_hora.grid(row=2, column=1, padx=10, pady=5)

# Botones

tk.Button(self, text="Calcular", command=self.calcular).grid(row=3, column=0, padx=10, pady=10)

tk.Button(self, text="Borrar", command=self.borrar).grid(row=3, column=1, padx=10, pady=10)

tk.Button(self, text="Salir", command=self.salir).grid(row=4, column=0, columnspan=2, padx=10, pady=10)

# Campo de salida

tk.Label(self, text="Respuesta:").grid(row=5, column=0, padx=10, pady=5)

self.txt\_respuesta = tk.Entry(self, state="readonly")

self.txt\_respuesta.grid(row=5, column=1, padx=10, pady=5)

def calcular(self):

try:

nombre = self.txt\_nombre.get()

horas\_trabajadas = float(self.txt\_horas\_mes.get())

valor\_hora = float(self.txt\_valor\_hora.get())

nomina = CalculoNomina(horas\_trabajadas, valor\_hora, nombre)

respuesta = nomina.calcular\_nomina()

# Mostrar respuesta en el campo de texto

self.txt\_respuesta.config(state="normal")

self.txt\_respuesta.delete(0, tk.END)

self.txt\_respuesta.insert(0, respuesta)

self.txt\_respuesta.config(state="readonly")

except ValueError:

messagebox.showerror("Error", "Por favor ingresa valores numéricos válidos para las horas trabajadas y el valor por hora.")

def borrar(self):

self.txt\_nombre.delete(0, tk.END)

self.txt\_horas\_mes.delete(0, tk.END)

self.txt\_valor\_hora.delete(0, tk.END)

self.txt\_respuesta.config(state="normal")

self.txt\_respuesta.delete(0, tk.END)

self.txt\_respuesta.config(state="readonly")

def salir(self):

self.destroy()

if \_\_name\_\_ == "\_\_main\_\_":

app = Ventana()

app.mainloop()

* 23. Dados los valores A, B y C que son los parámetros de una ecuación de segundo grado, elaborar un algoritmo para hallar las posibles soluciones de dicha ecuación.

Código:

import math

class CalculoEcuacion:

def \_\_init\_\_(self, a, b, c):

self.valor\_a = a

self.valor\_b = b

self.valor\_c = c

def calcular\_solucion\_ecuacion(self):

discriminante = self.calcular\_discriminante()

if discriminante > 0:

x1 = (-self.valor\_b + math.sqrt(discriminante)) / (2 \* self.valor\_a)

x2 = (-self.valor\_b - math.sqrt(discriminante)) / (2 \* self.valor\_a)

return f"Las soluciones son: {x1:.2f} y {x2:.2f}"

elif discriminante == 0:

x = -self.valor\_b / (2 \* self.valor\_a)

return f"La solución es: {x:.2f}"

else:

return "No tiene soluciones reales"

def calcular\_discriminante(self):

return (self.valor\_b \*\* 2) - (4 \* self.valor\_a \* self.valor\_c)

#INTERFAZ GRÁFICA:

import tkinter as tk

from tkinter import messagebox

class Ventana(tk.Tk):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.title("Cálculo de Ecuación Cuadrática")

# Campos de entrada para los valores A, B y C

tk.Label(self, text="Valor A:").grid(row=0, column=0, padx=10, pady=5)

self.txt\_a = tk.Entry(self)

self.txt\_a.grid(row=0, column=1, padx=10, pady=5)

tk.Label(self, text="Valor B:").grid(row=1, column=0, padx=10, pady=5)

self.txt\_b = tk.Entry(self)

self.txt\_b.grid(row=1, column=1, padx=10, pady=5)

tk.Label(self, text="Valor C:").grid(row=2, column=0, padx=10, pady=5)

self.txt\_c = tk.Entry(self)

self.txt\_c.grid(row=2, column=1, padx=10, pady=5)

# Botones

tk.Button(self, text="Calcular", command=self.calcular).grid(row=3, column=0, padx=10, pady=10)

tk.Button(self, text="Borrar", command=self.borrar).grid(row=3, column=1, padx=10, pady=10)

tk.Button(self, text="Salir", command=self.salir).grid(row=4, column=0, columnspan=2, padx=10, pady=10)

# Campo de salida para mostrar el resultado

tk.Label(self, text="Resultado:").grid(row=5, column=0, padx=10, pady=5)

self.txt\_resultado = tk.Entry(self, state="readonly", width=50) # Se agrega width=50

self.txt\_resultado.grid(row=5, column=1, padx=10, pady=5)

def calcular(self):

try:

a = float(self.txt\_a.get())

b = float(self.txt\_b.get())

c = float(self.txt\_c.get())

if a == 0:

messagebox.showerror("Error", "El valor de 'A' no puede ser cero en una ecuación cuadrática.")

return

ecuacion = CalculoEcuacion(a, b, c)

resultado = ecuacion.calcular\_solucion\_ecuacion()

# Mostrar el resultado

self.txt\_resultado.config(state="normal")

self.txt\_resultado.delete(0, tk.END)

self.txt\_resultado.insert(0, resultado)

self.txt\_resultado.config(state="readonly")

except ValueError:

messagebox.showerror("Error", "Por favor ingresa valores numéricos válidos.")

def borrar(self):

self.txt\_a.delete(0, tk.END)

self.txt\_b.delete(0, tk.END)

self.txt\_c.delete(0, tk.END)

self.txt\_resultado.config(state="normal")

self.txt\_resultado.delete(0, tk.END)

self.txt\_resultado.config(state="readonly")

def salir(self):

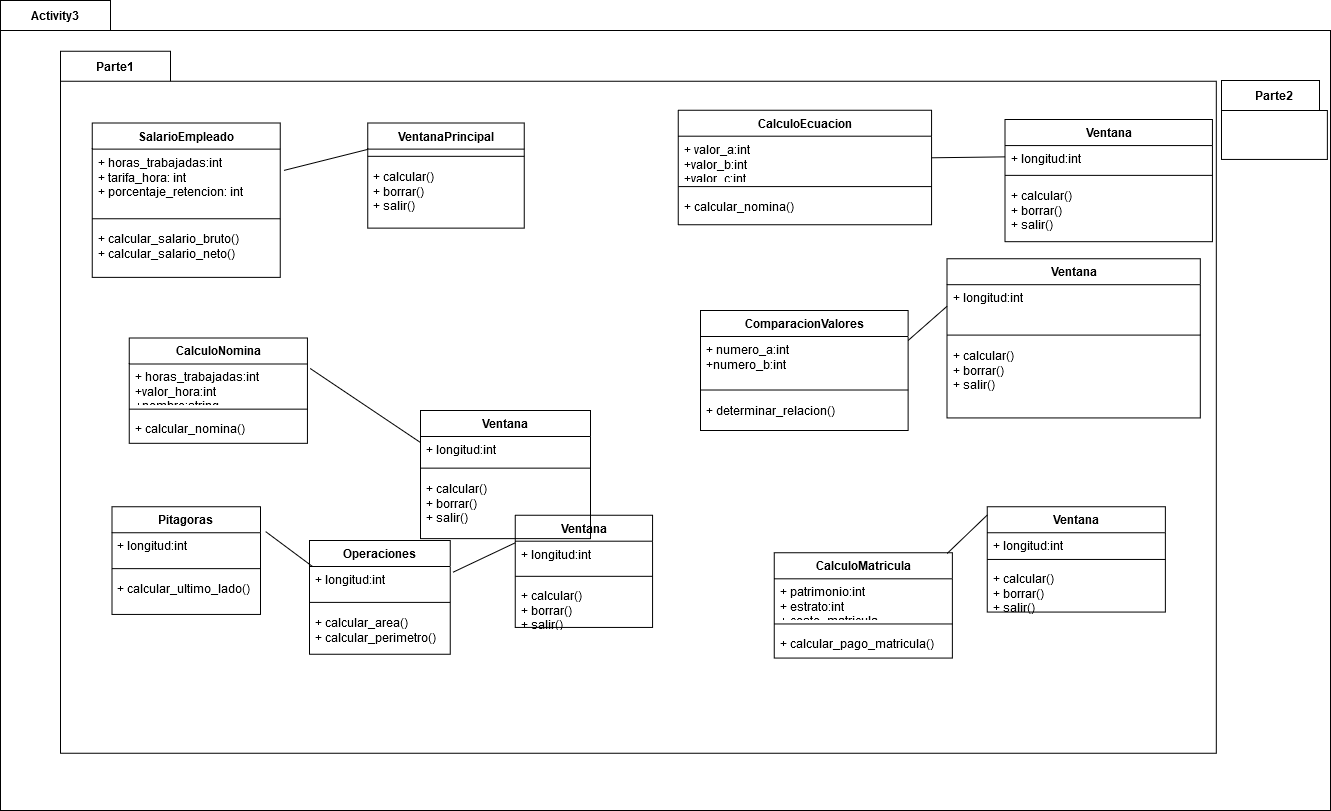
self.destroy()

if \_\_name\_\_ == "\_\_main\_\_":

app = Ventana()

app.mainloop()

Diagrama UML



PARTE 2

Código fuente: <https://github.com/danielzp2000/actividad3PooUnalMed/tree/660e38e25dcc489a73b455f4fb8b64a8c92cb91f/Parte%202>

Se requiere un programa que modele varias figuras geométricas: el círculo, el rectángulo, el cuadrado y el triángulo rectángulo.

* El círculo tiene como atributo su radio en centímetros.
* El rectángulo, su base y altura en centímetros.
* El cuadrado, la longitud de sus lados en centímetros.
* El triángulo, su base y altura en centímetros.

Se requieren métodos para determinar el área y el perímetro de cada figura geométrica. Además, para el triángulo rectángulo se requiere:

* Un método que calcule la hipotenusa del rectángulo.
* Un método para determinar qué tipo de triángulo es:
  + Equilátero: todos sus lados son iguales.
  + Isósceles: tiene dos lados iguales.
  + Escaleno: todos sus lados son diferentes.

Círculo

import math

class Circulo:

def \_\_init\_\_(self, radio):

self.radio = radio

def calcular\_area(self):

return math.pi \* self.radio \*\* 2

def calcular\_perimetro(self):

return 2 \* math.pi \* self.radio

#INTERFAZ GRÁFICA

import tkinter as tk

from tkinter import messagebox

class InterfazCirculo(tk.Tk):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.title("Cálculo de Círculo")

# Campo para ingresar el radio

tk.Label(self, text="Radio:").grid(row=0, column=0, padx=10, pady=10)

self.txt\_radio = tk.Entry(self)

self.txt\_radio.grid(row=0, column=1, padx=10, pady=10)

# Botones

tk.Button(self, text="Calcular", command=self.calcular).grid(row=1, column=0, padx=10, pady=10)

tk.Button(self, text="Limpiar", command=self.limpiar).grid(row=1, column=1, padx=10, pady=10)

tk.Button(self, text="Salir", command=self.salir).grid(row=2, column=0, columnspan=2, padx=10, pady=10)

# Campos de salida para área y perímetro

tk.Label(self, text="Área:").grid(row=3, column=0, padx=10, pady=10)

self.txt\_area = tk.Entry(self, state="readonly")

self.txt\_area.grid(row=3, column=1, padx=10, pady=10)

tk.Label(self, text="Perímetro:").grid(row=4, column=0, padx=10, pady=10)

self.txt\_perimetro = tk.Entry(self, state="readonly")

self.txt\_perimetro.grid(row=4, column=1, padx=10, pady=10)

def calcular(self):

try:

radio = float(self.txt\_radio.get())

if radio < 0:

messagebox.showerror("Error", "El radio no puede ser negativo.")

return

circulo = Circulo(radio)

area = circulo.calcular\_area()

perimetro = circulo.calcular\_perimetro()

# Mostrar resultados

self.txt\_area.config(state="normal")

self.txt\_area.delete(0, tk.END)

self.txt\_area.insert(0, f"{area:.2f}")

self.txt\_area.config(state="readonly")

self.txt\_perimetro.config(state="normal")

self.txt\_perimetro.delete(0, tk.END)

self.txt\_perimetro.insert(0, f"{perimetro:.2f}")

self.txt\_perimetro.config(state="readonly")

except ValueError:

messagebox.showerror("Error", "Por favor ingresa un valor numérico válido para el radio.")

def limpiar(self):

self.txt\_radio.delete(0, tk.END)

self.txt\_area.config(state="normal")

self.txt\_area.delete(0, tk.END)

self.txt\_area.config(state="readonly")

self.txt\_perimetro.config(state="normal")

self.txt\_perimetro.delete(0, tk.END)

self.txt\_perimetro.config(state="readonly")

def salir(self):

self.destroy()

if \_\_name\_\_ == "\_\_main\_\_":

app = InterfazCirculo()

app.mainloop()

Cuadrado

class Cuadrado:

def \_\_init\_\_(self, lado):

self.lado = lado

def calcular\_area(self):

return self.lado \*\* 2

def calcular\_perimetro(self):

return 4 \* self.lado

#INTERFEZ GRÁFICA

import tkinter as tk

from tkinter import messagebox

class InterfazCuadrado(tk.Tk):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.title("Cálculo de Cuadrado")

# Campo para ingresar el lado

tk.Label(self, text="Lado:").grid(row=0, column=0, padx=10, pady=10)

self.txt\_lado = tk.Entry(self)

self.txt\_lado.grid(row=0, column=1, padx=10, pady=10)

# Botones

tk.Button(self, text="Calcular", command=self.calcular).grid(row=1, column=0, padx=10, pady=10)

tk.Button(self, text="Limpiar", command=self.limpiar).grid(row=1, column=1, padx=10, pady=10)

tk.Button(self, text="Salir", command=self.salir).grid(row=2, column=0, columnspan=2, padx=10, pady=10)

# Campos de salida para área y perímetro

tk.Label(self, text="Área:").grid(row=3, column=0, padx=10, pady=10)

self.txt\_area = tk.Entry(self, state="readonly")

self.txt\_area.grid(row=3, column=1, padx=10, pady=10)

tk.Label(self, text="Perímetro:").grid(row=4, column=0, padx=10, pady=10)

self.txt\_perimetro = tk.Entry(self, state="readonly")

self.txt\_perimetro.grid(row=4, column=1, padx=10, pady=10)

def calcular(self):

try:

lado = float(self.txt\_lado.get())

if lado < 0:

messagebox.showerror("Error", "El lado no puede ser negativo.")

return

cuadrado = Cuadrado(lado)

area = cuadrado.calcular\_area()

perimetro = cuadrado.calcular\_perimetro()

# Mostrar resultados

self.txt\_area.config(state="normal")

self.txt\_area.delete(0, tk.END)

self.txt\_area.insert(0, f"{area:.2f}")

self.txt\_area.config(state="readonly")

self.txt\_perimetro.config(state="normal")

self.txt\_perimetro.delete(0, tk.END)

self.txt\_perimetro.insert(0, f"{perimetro:.2f}")

self.txt\_perimetro.config(state="readonly")

except ValueError:

messagebox.showerror("Error", "Por favor ingresa un valor numérico válido para el lado.")

def limpiar(self):

self.txt\_lado.delete(0, tk.END)

self.txt\_area.config(state="normal")

self.txt\_area.delete(0, tk.END)

self.txt\_area.config(state="readonly")

self.txt\_perimetro.config(state="normal")

self.txt\_perimetro.delete(0, tk.END)

self.txt\_perimetro.config(state="readonly")

def salir(self):

self.destroy()

if \_\_name\_\_ == "\_\_main\_\_":

app = InterfazCuadrado()

app.mainloop()

Rectángulo

class Rectangulo:

def \_\_init\_\_(self, base, altura):

self.base = base

self.altura = altura

def calcular\_area(self):

return self.base \* self.altura

def calcular\_perimetro(self):

return (2 \* self.base) + (2 \* self.altura)

# InTERFAZ GRÁFICA

import tkinter as tk

from tkinter import messagebox

class InterfazRectangulo(tk.Tk):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.title("Cálculo de Rectángulo")

# Campos para ingresar la base y la altura

tk.Label(self, text="Base:").grid(row=0, column=0, padx=10, pady=10)

self.txt\_base = tk.Entry(self)

self.txt\_base.grid(row=0, column=1, padx=10, pady=10)

tk.Label(self, text="Altura:").grid(row=1, column=0, padx=10, pady=10)

self.txt\_altura = tk.Entry(self)

self.txt\_altura.grid(row=1, column=1, padx=10, pady=10)

# Botones

tk.Button(self, text="Calcular", command=self.calcular).grid(row=2, column=0, padx=10, pady=10)

tk.Button(self, text="Limpiar", command=self.limpiar).grid(row=2, column=1, padx=10, pady=10)

tk.Button(self, text="Salir", command=self.salir).grid(row=3, column=0, columnspan=2, padx=10, pady=10)

# Campos de salida para área y perímetro

tk.Label(self, text="Área:").grid(row=4, column=0, padx=10, pady=10)

self.txt\_area = tk.Entry(self, state="readonly")

self.txt\_area.grid(row=4, column=1, padx=10, pady=10)

tk.Label(self, text="Perímetro:").grid(row=5, column=0, padx=10, pady=10)

self.txt\_perimetro = tk.Entry(self, state="readonly")

self.txt\_perimetro.grid(row=5, column=1, padx=10, pady=10)

def calcular(self):

try:

base = float(self.txt\_base.get())

altura = float(self.txt\_altura.get())

if base < 0 or altura < 0:

messagebox.showerror("Error", "La base y la altura deben ser valores positivos.")

return

rectangulo = Rectangulo(base, altura)

area = rectangulo.calcular\_area()

perimetro = rectangulo.calcular\_perimetro()

# Mostrar resultados

self.txt\_area.config(state="normal")

self.txt\_area.delete(0, tk.END)

self.txt\_area.insert(0, f"{area:.2f}")

self.txt\_area.config(state="readonly")

self.txt\_perimetro.config(state="normal")

self.txt\_perimetro.delete(0, tk.END)

self.txt\_perimetro.insert(0, f"{perimetro:.2f}")

self.txt\_perimetro.config(state="readonly")

except ValueError:

messagebox.showerror("Error", "Por favor ingresa valores numéricos válidos para la base y la altura.")

def limpiar(self):

self.txt\_base.delete(0, tk.END)

self.txt\_altura.delete(0, tk.END)

self.txt\_area.config(state="normal")

self.txt\_area.delete(0, tk.END)

self.txt\_area.config(state="readonly")

self.txt\_perimetro.config(state="normal")

self.txt\_perimetro.delete(0, tk.END)

self.txt\_perimetro.config(state="readonly")

def salir(self):

self.destroy()

if \_\_name\_\_ == "\_\_main\_\_":

app = InterfazRectangulo()

app.mainloop()

Triangulo

import math

class TrianguloRectangulo:

def \_\_init\_\_(self, base, altura):

self.base = base

self.altura = altura

def calcular\_area(self):

return (self.base \* self.altura) / 2

def calcular\_perimetro(self):

return self.base + self.altura + self.calcular\_hipotenusa()

def calcular\_hipotenusa(self):

return math.sqrt(self.base \*\* 2 + self.altura \*\* 2)

def determinar\_tipo\_triangulo(self):

hipotenusa = self.calcular\_hipotenusa()

if self.base == self.altura == hipotenusa:

return "Equilátero"

elif self.base != self.altura and self.base != hipotenusa and self.altura != hipotenusa:

return "Escaleno"

else:

return "Isósceles"

#Interfaz Gráfica:

import tkinter as tk

from tkinter import messagebox

class InterfazTriangulo(tk.Tk):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.title("Cálculo de Triángulo Rectángulo")

# Campos para ingresar la base y la altura

tk.Label(self, text="Base:").grid(row=0, column=0, padx=10, pady=10)

self.txt\_base = tk.Entry(self)

self.txt\_base.grid(row=0, column=1, padx=10, pady=10)

tk.Label(self, text="Altura:").grid(row=1, column=0, padx=10, pady=10)

self.txt\_altura = tk.Entry(self)

self.txt\_altura.grid(row=1, column=1, padx=10, pady=10)

# Botones

tk.Button(self, text="Calcular", command=self.calcular).grid(row=2, column=0, padx=10, pady=10)

tk.Button(self, text="Limpiar", command=self.limpiar).grid(row=2, column=1, padx=10, pady=10)

tk.Button(self, text="Salir", command=self.salir).grid(row=3, column=0, columnspan=2, padx=10, pady=10)

# Campos de salida para área, perímetro, hipotenusa y tipo de triángulo

tk.Label(self, text="Área:").grid(row=4, column=0, padx=10, pady=10)

self.txt\_area = tk.Entry(self, state="readonly")

self.txt\_area.grid(row=4, column=1, padx=10, pady=10)

tk.Label(self, text="Perímetro:").grid(row=5, column=0, padx=10, pady=10)

self.txt\_perimetro = tk.Entry(self, state="readonly")

self.txt\_perimetro.grid(row=5, column=1, padx=10, pady=10)

tk.Label(self, text="Hipotenusa:").grid(row=6, column=0, padx=10, pady=10)

self.txt\_hipotenusa = tk.Entry(self, state="readonly")

self.txt\_hipotenusa.grid(row=6, column=1, padx=10, pady=10)

tk.Label(self, text="Tipo de Triángulo:").grid(row=7, column=0, padx=10, pady=10)

self.txt\_tipo\_triangulo = tk.Entry(self, state="readonly")

self.txt\_tipo\_triangulo.grid(row=7, column=1, padx=10, pady=10)

def calcular(self):

try:

base = float(self.txt\_base.get())

altura = float(self.txt\_altura.get())

if base <= 0 or altura <= 0:

messagebox.showerror("Error", "La base y la altura deben ser mayores que cero.")

return

triangulo = TrianguloRectangulo(base, altura)

area = triangulo.calcular\_area()

perimetro = triangulo.calcular\_perimetro()

hipotenusa = triangulo.calcular\_hipotenusa()

tipo\_triangulo = triangulo.determinar\_tipo\_triangulo()

# Mostrar resultados

self.txt\_area.config(state="normal")

self.txt\_area.delete(0, tk.END)

self.txt\_area.insert(0, f"{area:.2f}")

self.txt\_area.config(state="readonly")

self.txt\_perimetro.config(state="normal")

self.txt\_perimetro.delete(0, tk.END)

self.txt\_perimetro.insert(0, f"{perimetro:.2f}")

self.txt\_perimetro.config(state="readonly")

self.txt\_hipotenusa.config(state="normal")

self.txt\_hipotenusa.delete(0, tk.END)

self.txt\_hipotenusa.insert(0, f"{hipotenusa:.2f}")

self.txt\_hipotenusa.config(state="readonly")

self.txt\_tipo\_triangulo.config(state="normal")

self.txt\_tipo\_triangulo.delete(0, tk.END)

self.txt\_tipo\_triangulo.insert(0, tipo\_triangulo)

self.txt\_tipo\_triangulo.config(state="readonly")

except ValueError:

messagebox.showerror("Error", "Por favor ingresa valores numéricos válidos.")

def limpiar(self):

self.txt\_base.delete(0, tk.END)

self.txt\_altura.delete(0, tk.END)

self.txt\_area.config(state="normal")

self.txt\_area.delete(0, tk.END)

self.txt\_area.config(state="readonly")

self.txt\_perimetro.config(state="normal")

self.txt\_perimetro.delete(0, tk.END)

self.txt\_perimetro.config(state="readonly")

self.txt\_hipotenusa.config(state="normal")

self.txt\_hipotenusa.delete(0, tk.END)

self.txt\_hipotenusa.config(state="readonly")

self.txt\_tipo\_triangulo.config(state="normal")

self.txt\_tipo\_triangulo.delete(0, tk.END)

self.txt\_tipo\_triangulo.config(state="readonly")

def salir(self):

self.destroy()

if \_\_name\_\_ == "\_\_main\_\_":

app = InterfazTriangulo()

app.mainloop()

Diagrama UML

